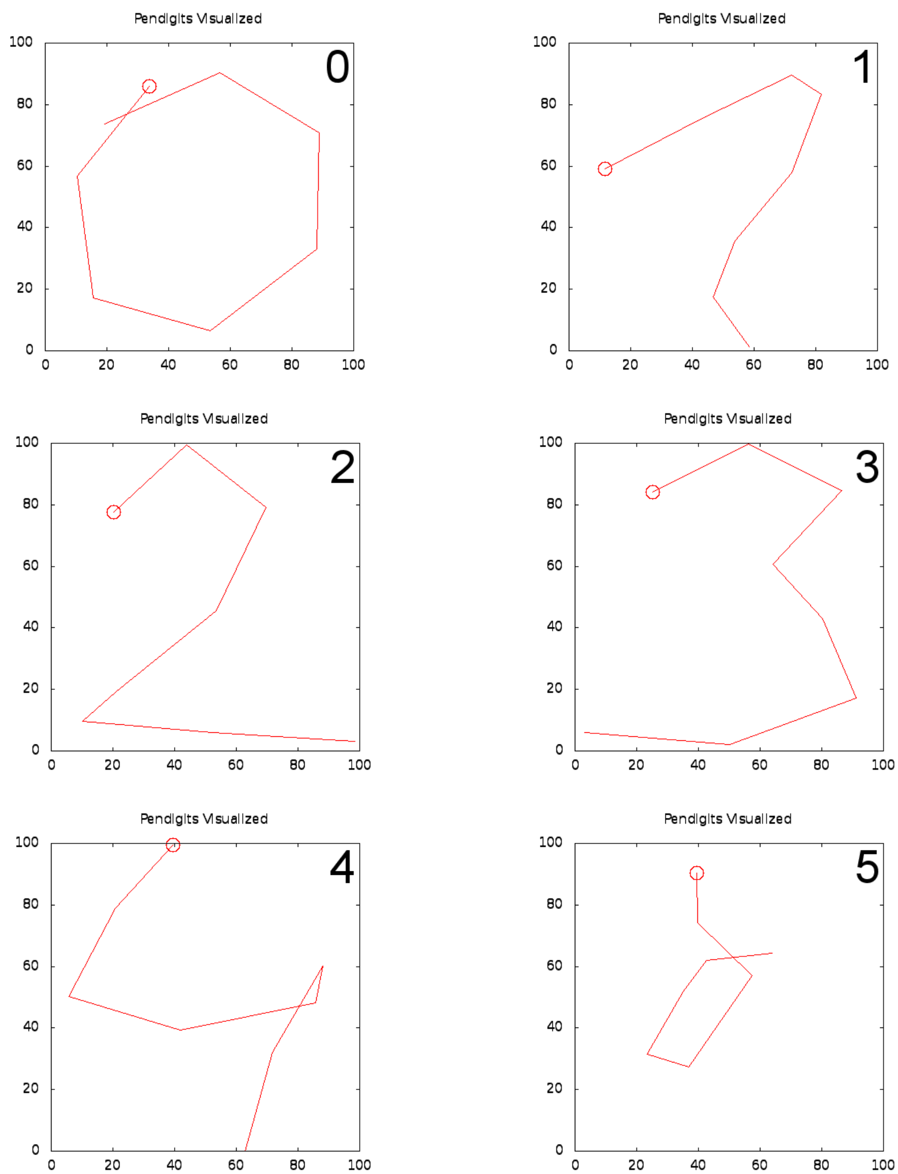
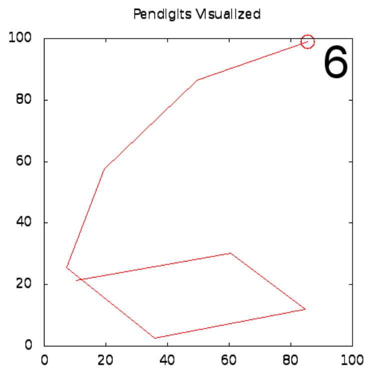
Mustererkennung - Aufgabenblatt 02

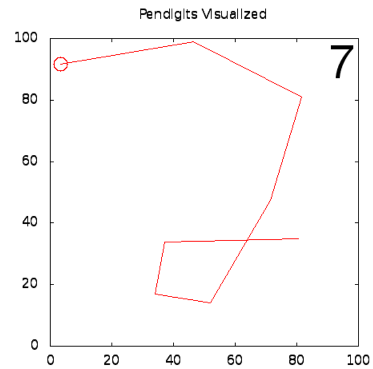
André Hacker und Dimitri Schachmann

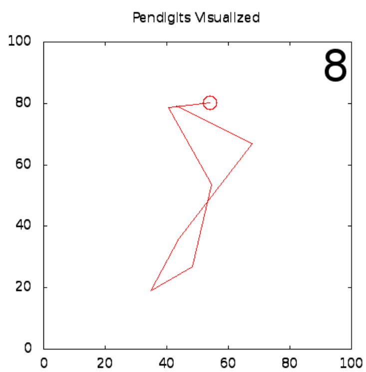
1. Visualisierung

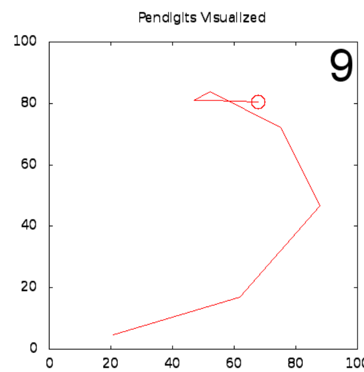
Wir haben mit Matlab die Mittlewerte der Daten berechnet und sie so wie beim ersten Übungsblatt mit gnuplott dargestellt.











2. Klassifikation

a) Mit cov, mean und mvnpdf

Wir haben die Klassifikation mittels der multivariaten Normalverteilung in Matlab implementiert. Zunächst haben wir das mit Hilfe der Matlab funktionen **cov**, **mean** und **mvnpdf** implementiert, was sehr einfach war:

```
% --------------
% Classifier using multivariant normal distribution
% --------------
function covar
  % Load input into (nx17) Matrix.
  % last column stores the class number
  tra = load('pendigits.tra');
  tes = load('pendigits.tes');

  % Iterate through all classes
  % After this loop, P will be a (nx10) matrix where
  % n = #test records and
  % P(i,c) = Probability of test-item i being in class c-1
  P = [];
  for i = 0:9
    %get all training-data for this class
    currentTrainingData = filterByClass(tra, i);
```

```
    % Compute mean and covariance matrix for this class
    [mu sigma] = meanAndCov(currentTrainingData);

    % Compute the probability density for all test-items
    % Add those as a column to P
    P = [P mvnpdf(tes(:,1:end-1), mu, sigma)];
  end

  % For each row get the index that has the highest value (probability)
  [maxValue maxIndex] = max(P,[],2);

  % Index begins with 1, our classes with 0
  maxIndex = maxIndex - 1;

  dlmwrite('recognition_results.mat', [maxIndex tes(:,end)], ' ');

end

% Input: (nx17)-Matrix with labled data (label in last column)
% Output: all rows having class c (without label column)
function r = filterByClass(data, c)
  r = data(ismember(data(:,end),c),1:end-1);
end

% Computes mean and covariance based on matrix with observations
% Input: rows with training data for one class
% Output: Mean (row vector) and covariance matrix (16x16)
function [m c] = meanAndCov(data)
  m = mean(data);
  c = cov(data);
  c = c + (0.0001 * eye(size(c)));
end
```

b) Ohne cov und mvpdf

Dann haben wir das ganze noch mal mit eigener Kovarianz, Mittelwert und Dichtefunktion implementiert:

```
% --------------
% Classifier using multivariant normal distribution
% SUCCESS RATE: 0.959
% --------------
function covar2
  % Load input into (nx17) Matrix.
  % last column stores the class number
  tra = load('pendigits.tra');
  tes = load('pendigits.tes');

  % Iterate through all classes
  % After this loop, P will be a (nx10) matrix where
  % n = #test records and
  % P(i,c) = Probability of test-item i being in class c-1
```

```
  P = [];
  for i = 0:9
    %get all training-data for this class
    train = filterByClass(tra, i);

    % Compute mean and covariance matrix for this class
    mu = getMean(train);
    sigma = getCovarMatrix(train, mu);

    % Compute the probability density for all test-items
    % Add those as a column to P
    %P = [P mvnpdf(tes(:,1:end-1), mu, sigma)];
    P = [P multivariateDensity(tes(:,1:end-1), mu, sigma)];
  end
  %disp(P(1:1,:));
  % For each row get the index that has the highest value (probability)
  [maxValue maxIndex] = max(P,[],2);

  % Index begins with 1, our classes with 0
  maxIndex = maxIndex - 1;

  dlmwrite('recognition_results.mat', [maxIndex tes(:,end)], ' ');

end

% Input: (nx17)-Matrix with labled data (label in last column)
% Output: all rows having class c (without label column)
function r = filterByClass(data, c)
  r = data(ismember(data(:,end),c),1:end-1);
end

% Computes mean based on matrix with observations
% Input: rows with training data for one class
% Output: Mean (row vector)
function m = getMean(data)
  m = 1/size(data,1) * sum(data);
end

% Computes covariance matrix based on matrix with observations
% Input: rows with training data for one class
% Output: Covariance matrix (16x16)
function c = getCovarMatrix(data, mu)
  c = zeros(16,16);
  mu=mu';
  for i=1:size(data,1)
    xi = data(i,:)';
    c = c + (xi-mu) * (xi-mu)';
  end

  % Normalize
  c = c / size(data,1);

  %c = cov(data); % it would be so simple;)
```

```
  % make some noise;)
  c = c + (0.0001 * eye(size(c)));
end

% Multivariate normal distribution density function (pdf)
% produces same output as mvnpdf, based on the formula in the rojas tutorial
function p = multivariateDensity(data, mu, sigma)
  mu = mu'; % We work with column vectors here
  normalize = 1/sqrt(det(2 * pi * sigma));
  p = zeros(size(data,1), 1);
  for i=1:size(data,1)
    cur = data(i,:)';
    p(i) = normalize * exp( -0.5 * (cur-mu)' * inv(sigma) * (cur-mu) );
  end
end
```

c) Erkennungsrate und Confusion Matrix

Die Erkennungsrate dies Algorithmus ist 95.9%, was wir für überraschend gut halten. Wenn man bedenkt, dass der Algorithmus aus dem 1. Übungsblatt (zumindest bei uns) eine viertel Stunde gelaufen ist und mit 97.4% nur unwesentlich besser war, während dieser in 1 Sekunde fertig ist.

	0	1	2	3	4	5	6	7	8	9
0	341	0	0	0	0	0	0	0	22	0
1	0	350	12	0	1	0	0	0	1	0
2	0	8	355	0	0	0	0	1	0	0
3	0	9	0	320	0	1	0	1	0	5
4	0	0	0	0	362	0	0	0	0	2
5	0	0	0	1	0	323	0	0	2	9
6	0	0	0	0	0	0	325	0	11	0
7	0	28	0	0	0	0	0	314	5	17
8	0	0	0	0	0	0	0	0	336	0
9	0	5	0	0	0	0	0	1	1	329

3. Code für die Visualisierung

Für die Visualisierung der Ziffern haben wir ein einfaches gnuplot Skript geschrieben:

```
set title "Pendigits Visualized"
set terminal png

set size square
set xrange [0:100]
set yrange [0:100]


filename = sprintf("images/mean%i.png",i)
set output filename
set multiplot
plot 'mean_digits.plt' index i using (95):(88):1 every ::::0 with labels notitle\
     font "Arial,44"
```

```
plot 'mean_digits.plt' index i using 1:2 every ::1 with lines notitle
plot 'mean_digits.plt' index i using 1:2 every ::1::1 with points notitle ps 3 pt 6
unset multiplot
set output
```